LA-UR-18-27195
Approved for public release; distribution is unlimited.

Title: Performance Study and Optimization of FleCSALE using Tabular Equation of State

Author(s): Payne, Patrick Charles
Stegmeier, Nicholas William
Lakshmiranganatha, Sumathi
Akhmetova, Dana
Mukherjee, Diptajyoti
Ouellet, Frederick

Intended for: General Presentation

Issued: 2018-07-31

National Nuclear Security Administration
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

Performance Study and Optimization of FIeCSALE using Tabular Equation of State

Dana Akhmetova
Sumathi Lakshmiranganatha
Diptajyoti Mukherjee
Frederick Ouellet
Patrick Payne
Nicholas Stegmeier

Los Alamos NATIONAL LABORATORY EST.1943

National Nuclear Security Administration
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

Objective

Explore new numerical and computer science approaches to efficiently use EOSPAC within the context of FIeCSALE and reduce the cost of equation of state (EOS) table queries

Motivation

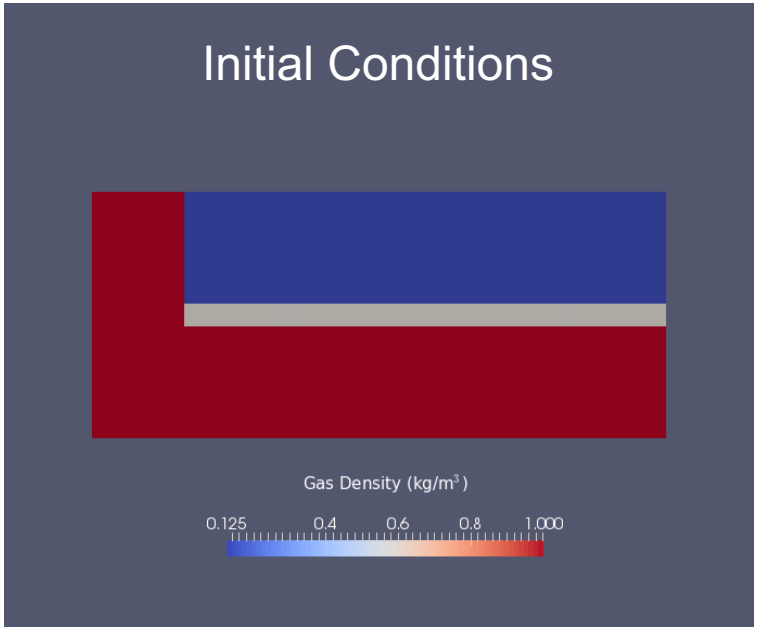
- Tabular equation of state (EOS) data are constantly queried by physics codes
- Queries can occur several times per time-step at each computational cell for each material that exists at a query location
- Queries regularly involve expensive interpolations of thermodynamic quantities
- Thermodynamic quantities are necessary for computation of additional physical quantities for the simulations
- Reducing the cost of tabular EOS queries provides an overall reduction in the solution time for many complex physics problems

FleCSALE, FleCSI, + EOSPAC

- **FleCSALE**
  - Continuum dynamics code, supports multi-phase fluids and tabular equation of state (EOS)
  - 2D/3D cell-centered Eulerian and Lagrangian solver
  - 3D FEM Lagrangian solver
- **FleCSI**
  - Compile-time configurable framework designed to support multi-physics application development
  - Supports an MPI and Legion backend
  - Primary structure is hierarchical, exposing low-level, mid-level, and high-level interfaces that are appropriate for different sets of users
- **EOSPAC**
  - Collection of interface routines
  - Used to access the SESAME data library
  - Can perform various data adjustments and interpolations on the SESAME data
  - Inherently serial execution

Triple Point Problem

- **Fluids simulation involving three different materials**
  - All materials are ideal gases
- **Problem decomposed into four regions**
  - Left-most and bottom regions: two different materials at the same density
  - Top region: third material at low density
  - Middle region: mixed composition & density
- **All materials can use tabular EOS data for the simulations.**



Bottlenecks and Strategies

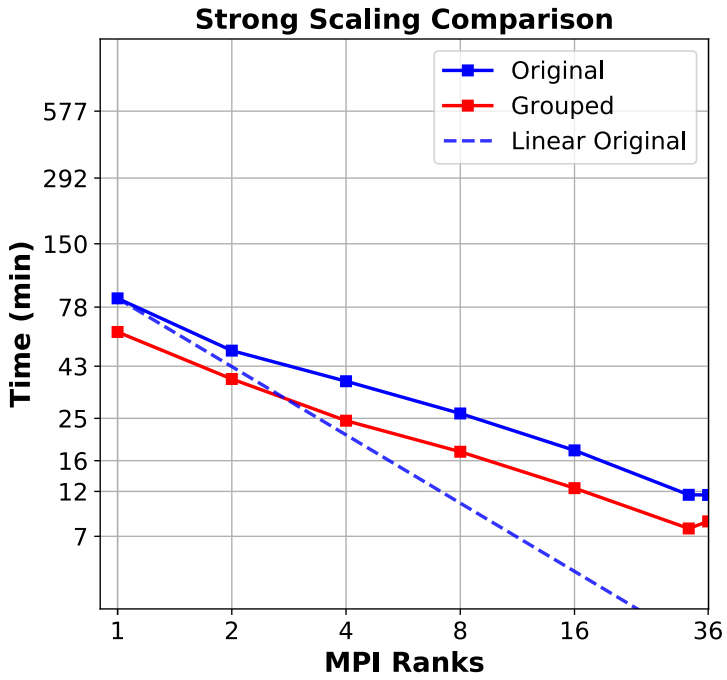
- **Initial Bottlenecks:**
  - Ghost Data Creation
  - Material Grouping
  - MPI Communication
  - Interpolation

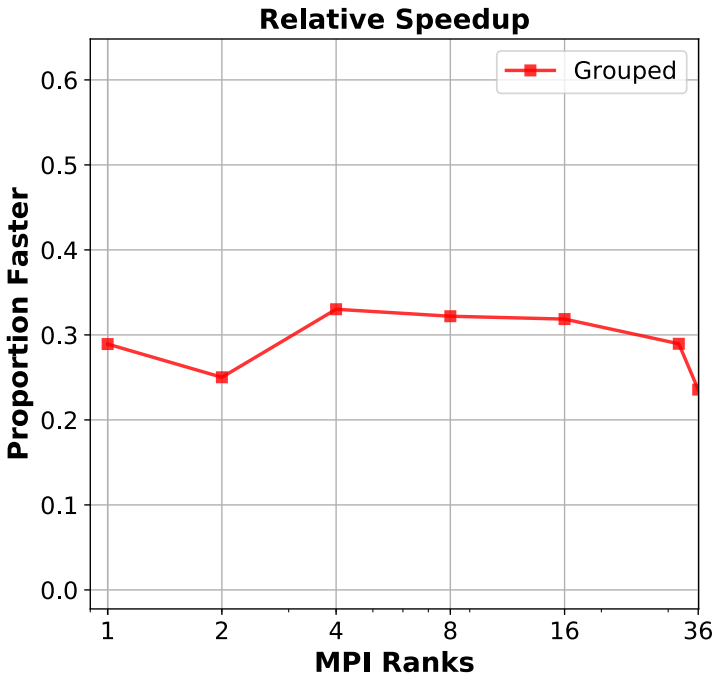
- **Explored Strategies:**
  - Software Engineering
  - MPI Communication Optimization --- sparse data
  - Hybrid Programming --- OpenMP
  - GPU Porting with different memory schemes
  - Machine Learning

Software Engineering

FleCSALE/EOSPAC Linking

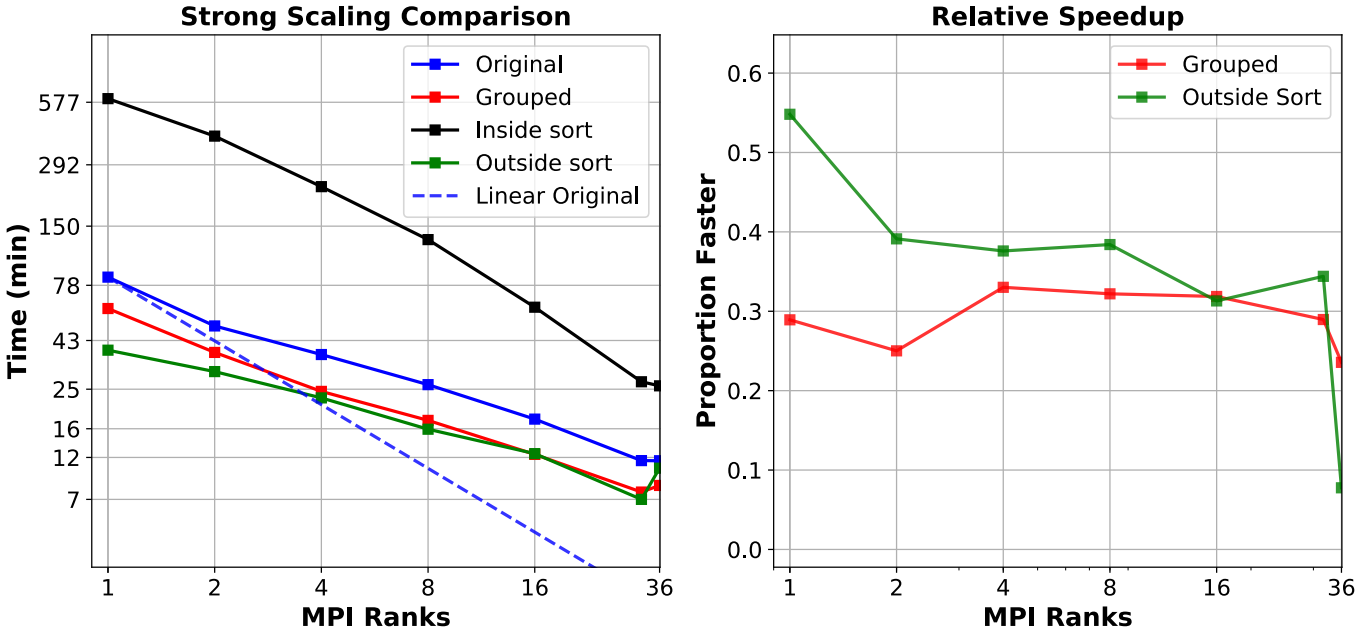
- Linking Optimizations:
  - Enabled the inversion/storing of modified Sesame tables at the initialization of FleCSALE
  - Reformatted FleCSALE to send common material cells to EOSPAC as groups





Sorting in EOSPAC's Search Algorithm

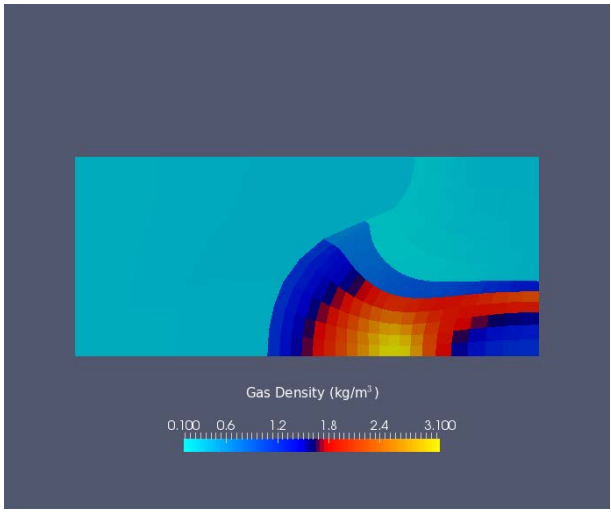
- Before interpolations, EOSPAC places a stencil on the material table → eos_srchdf()
- Approach uses an input to eos_srchdf() which gives the first point for the table sweep
- The indexes of the array of points to be interpolated are sorted before sweeping
- Two versions:
  - Embedded in EOSPAC
  - Inside of FleCSALE



Problem-Dependent Performance

- There is potential for enhanced performance from the index sorting which is problem dependent

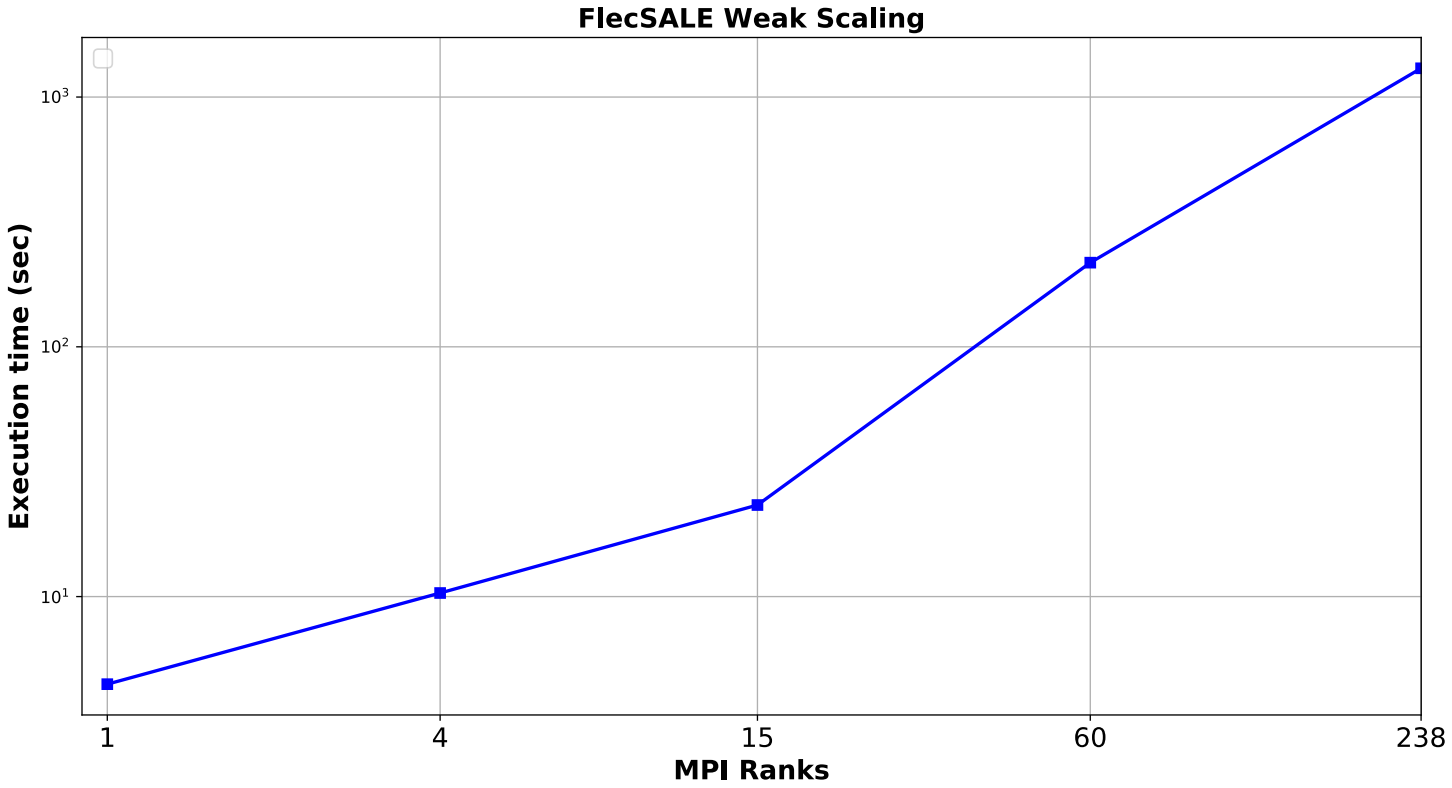
- Example: CDSS Scaling Problem (Triple Point)
  - Simulation density range = 0 to 3.5
  - Sesame table range = 0 to ~61,000
  - We only use roughly 20% of the table for this problem



- Problems with larger property variations are more likely to obtain better performance with the sorting approach

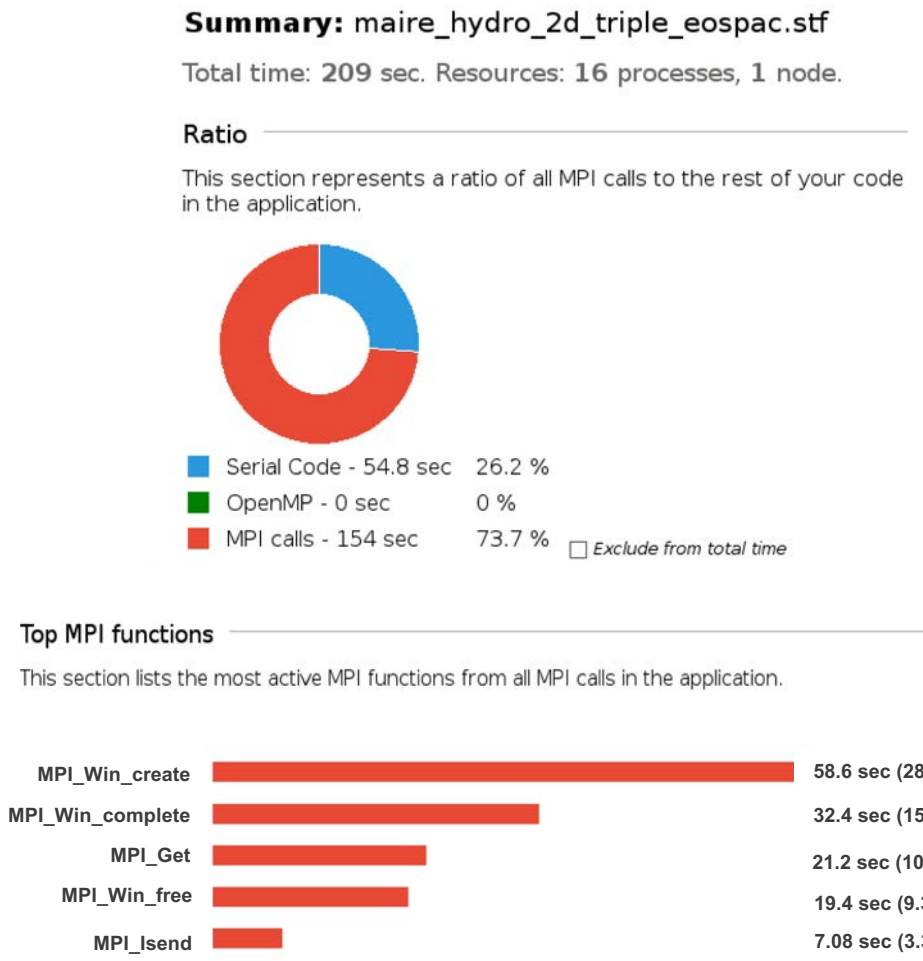
Sparse Data Optimization

Motivation


FlecSALE Weak Scaling
Execution time (sec)
10^3
10^2
10^1
1
4
15
60
238
MPI Ranks

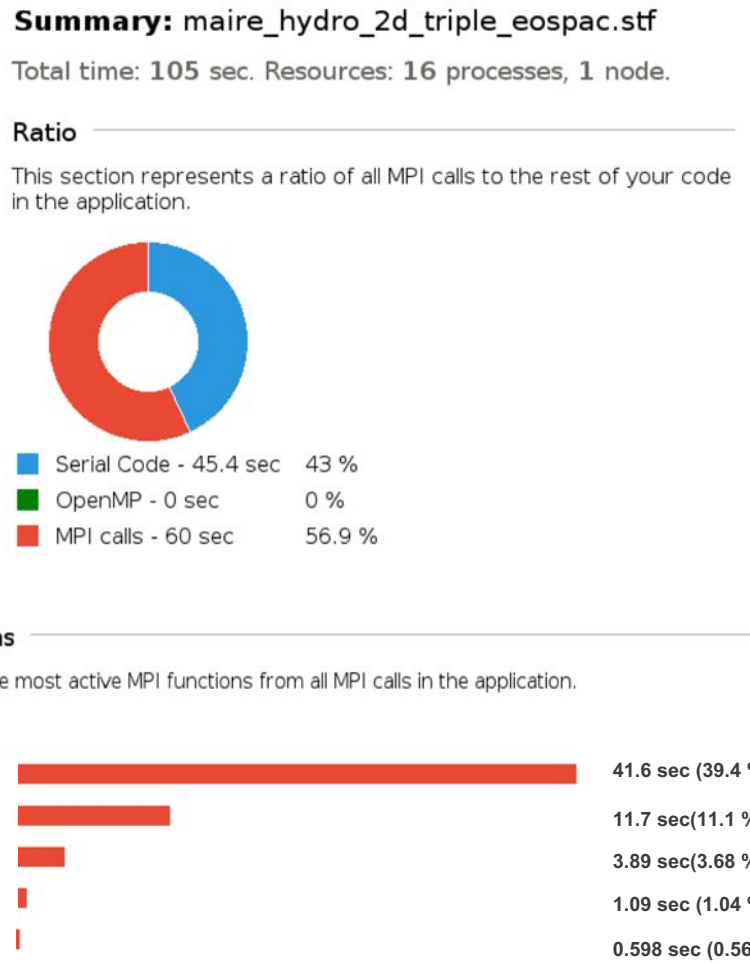
Sparse Data Optimization

- Weak scaling graph shows huge communication latency
- MPI backend in FleCSI uses One Sided Communication
- Tracing the application showed ~74% of the code was MPI communication calls
- **MPI_Win_create** calls was the most expensive with ~28% of the execution time



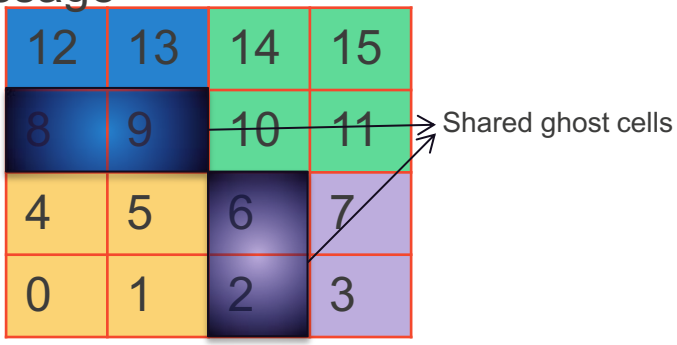
Sparse Data Optimization –Copy removal

- Unnecessary copies of data were eliminated
- Shared window creation was done only once with reduction of **MPI_Win_create** calls
- MPI Communications takes about ~57%
- **Performance Improvement**
  - Single node ~20%
  - Multi node ~40%



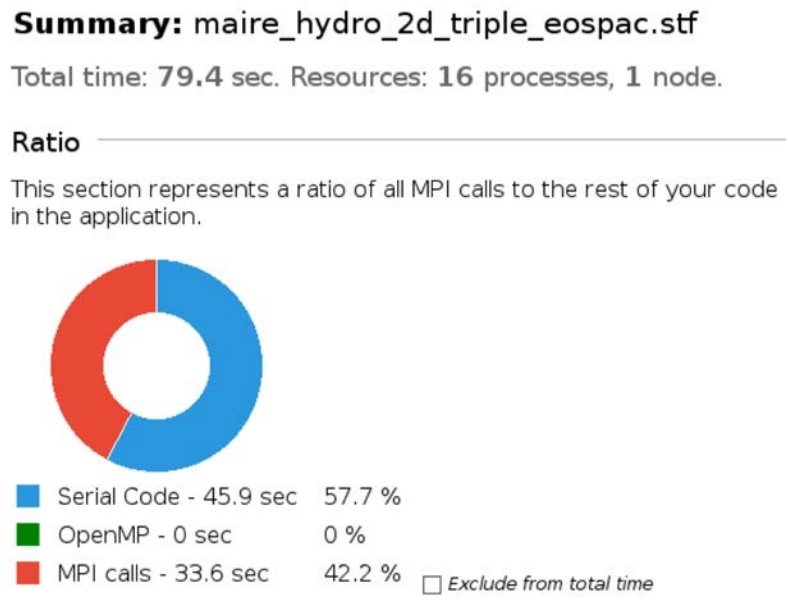
Sparse Data Optimization- MPI_Datatype

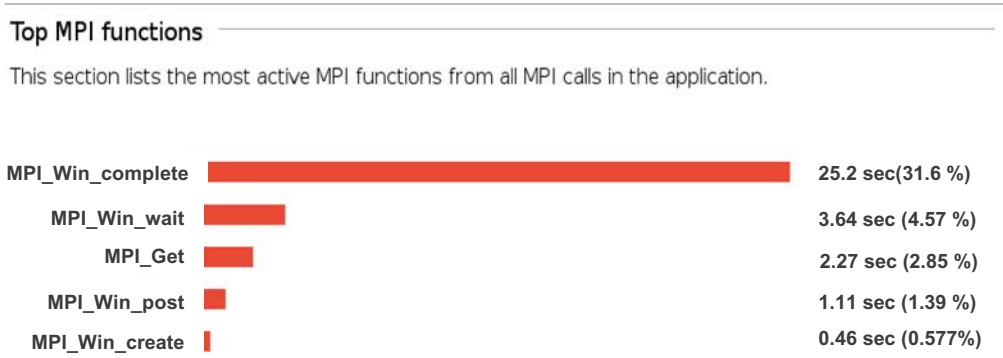
- **MPI_Datatype** is used for shared ghost cells to treat as a single message



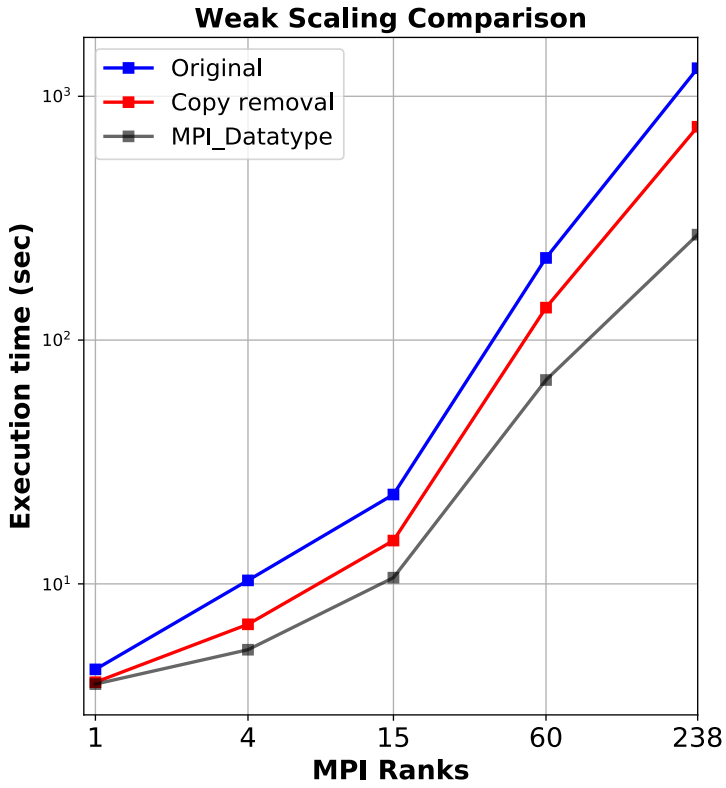
- **MPI_Get** and **MPI_Win_complete** calls reduced
- MPI Communication takes about 43%

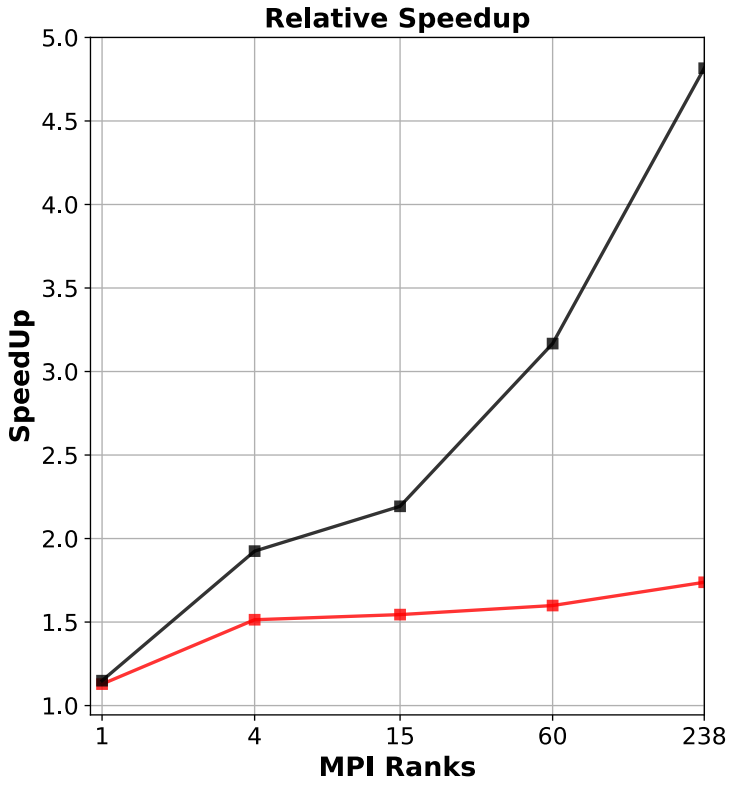
- **Performance Improvement**
  - Single node ~25%
  - Multi node ~60%





Performance Analysis



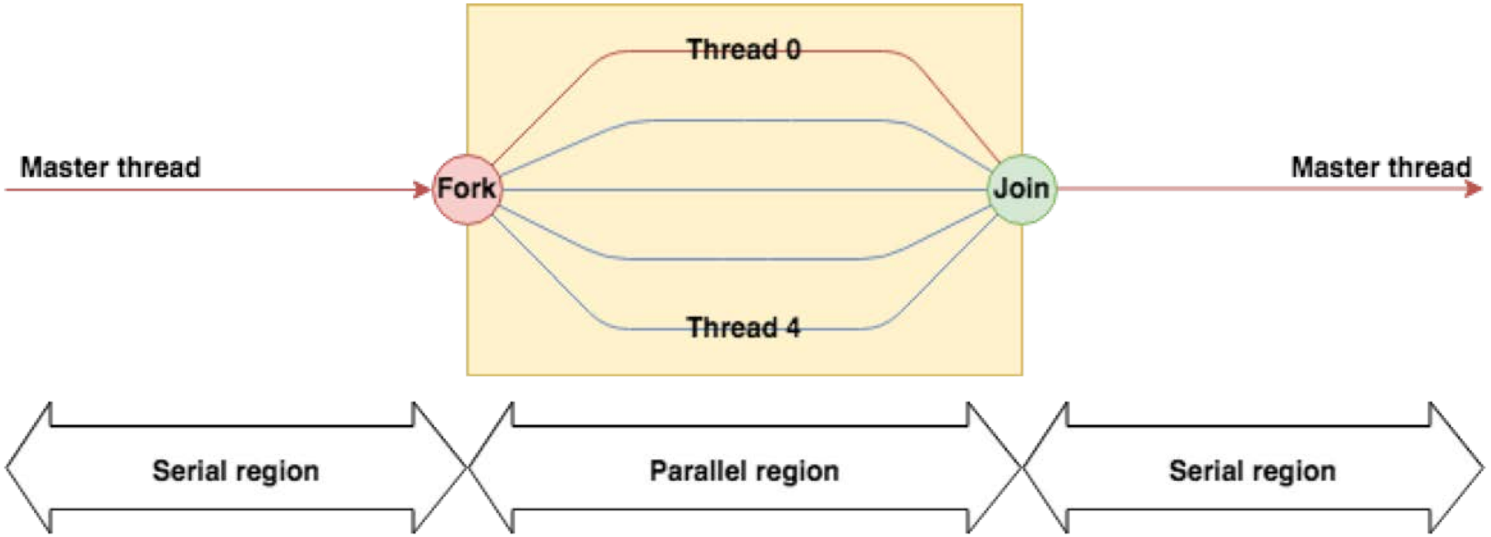


- Overall performance improvement ~80%
- Speedup of ~5x
- MPI_Win_complete is still expensive

Hybrid programming with OpenMP

Adding OpenMP in FleCSALE

- **Fork/join model:**

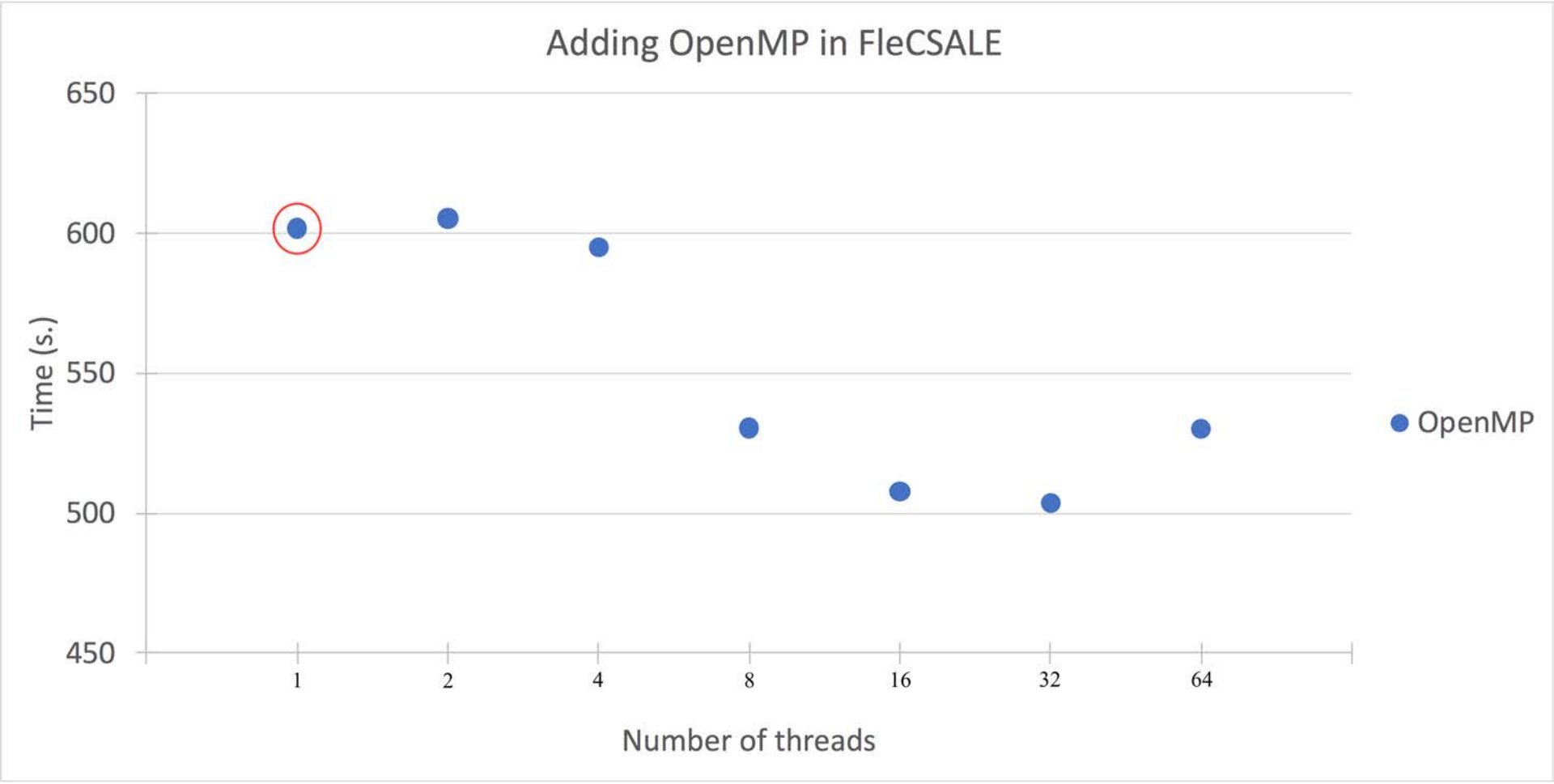


- **Two OpenMP approaches were investigated:**
  - tasking;
  - for-loop work-sharing.

Adding OpenMP in Flecsale

- **Changes done in:** maire_hydro/tasks.h
- **Mesh:** triple_200x81.g
- **Flags:** -g -O3 -fopenmp
- **Compilers:** gcc/7.3.0 and intel/18.0.2
- **MPI implementation:** mpich/3.2.1-intel_18.0.2
- **Platform:** Intel Haswell --- E5-2698 v3
  - Sockets --- 2
  - Cores per Socket --- 16
  - NUMA Domains --- 2

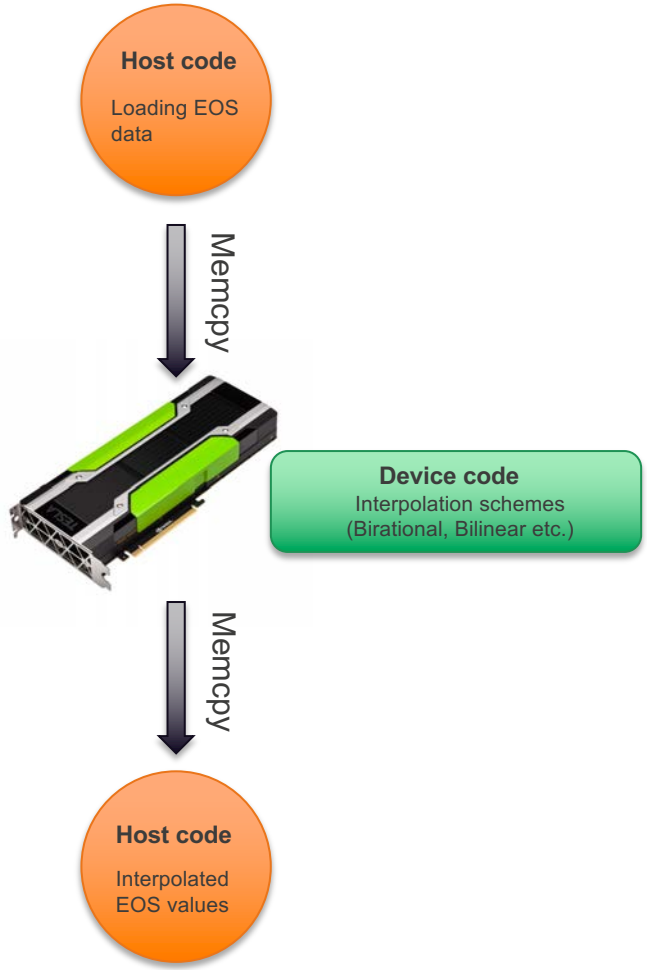
Performance results



GPU Porting for EOSPAC

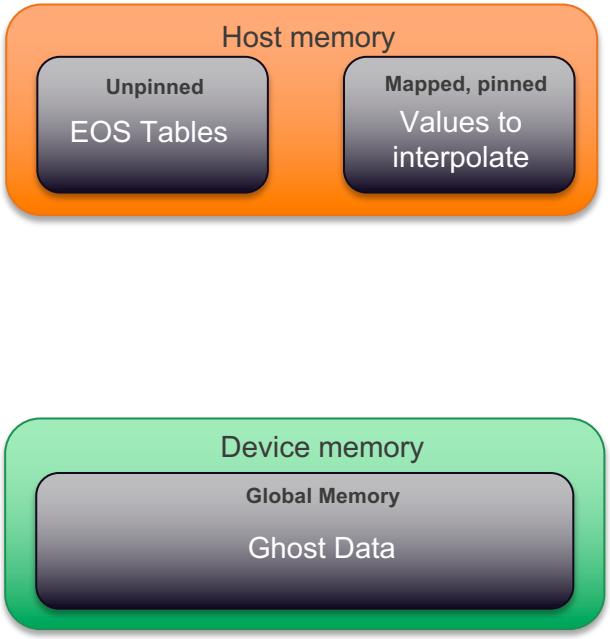
Motivation

- FleCSALE queries EOSPAC for interpolated EOS values.
- Profiling revealed Interpolation was costliest part of FleCSALE, ~15% of simulation.
- Can be parallelized easily.
- GPUs are next generation hardware for intensive computations.
- Interpolation is not memory intensive.



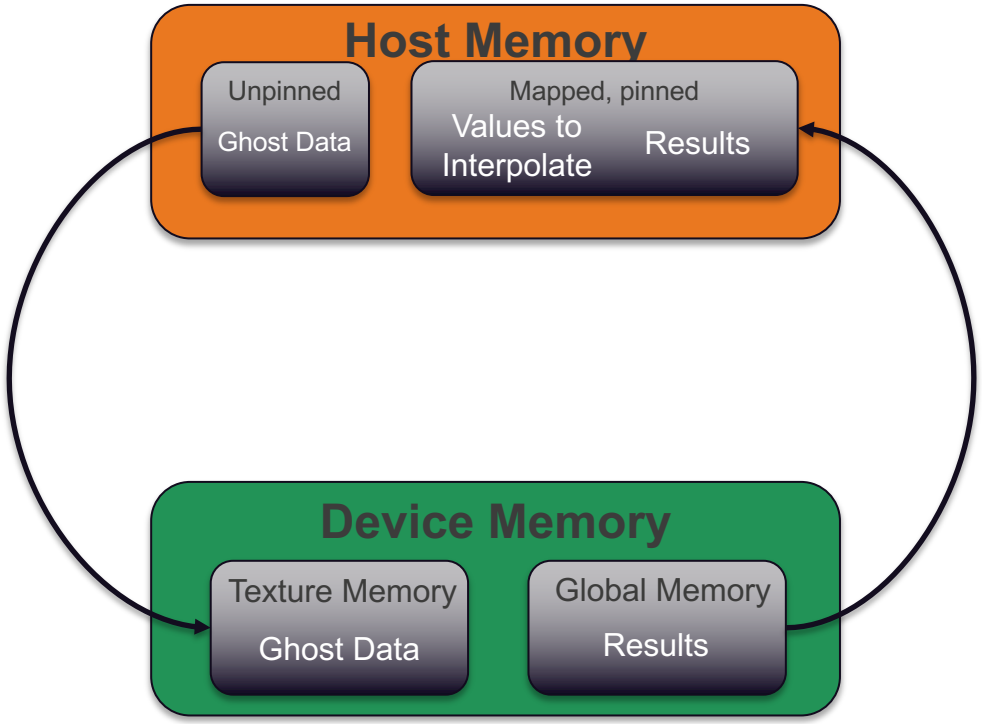
Methodology

- Wrapper around Birational Interpolation kernel and Bilinear Interpolation kernel that can be called inside EOSPAC.
- No changes to the public interfaces. Only need a separate library compilation.
- Interpolation algorithms are called multiple times within a single FleCSALE time-step.
- No Memcpy is necessary for the interpolated values.

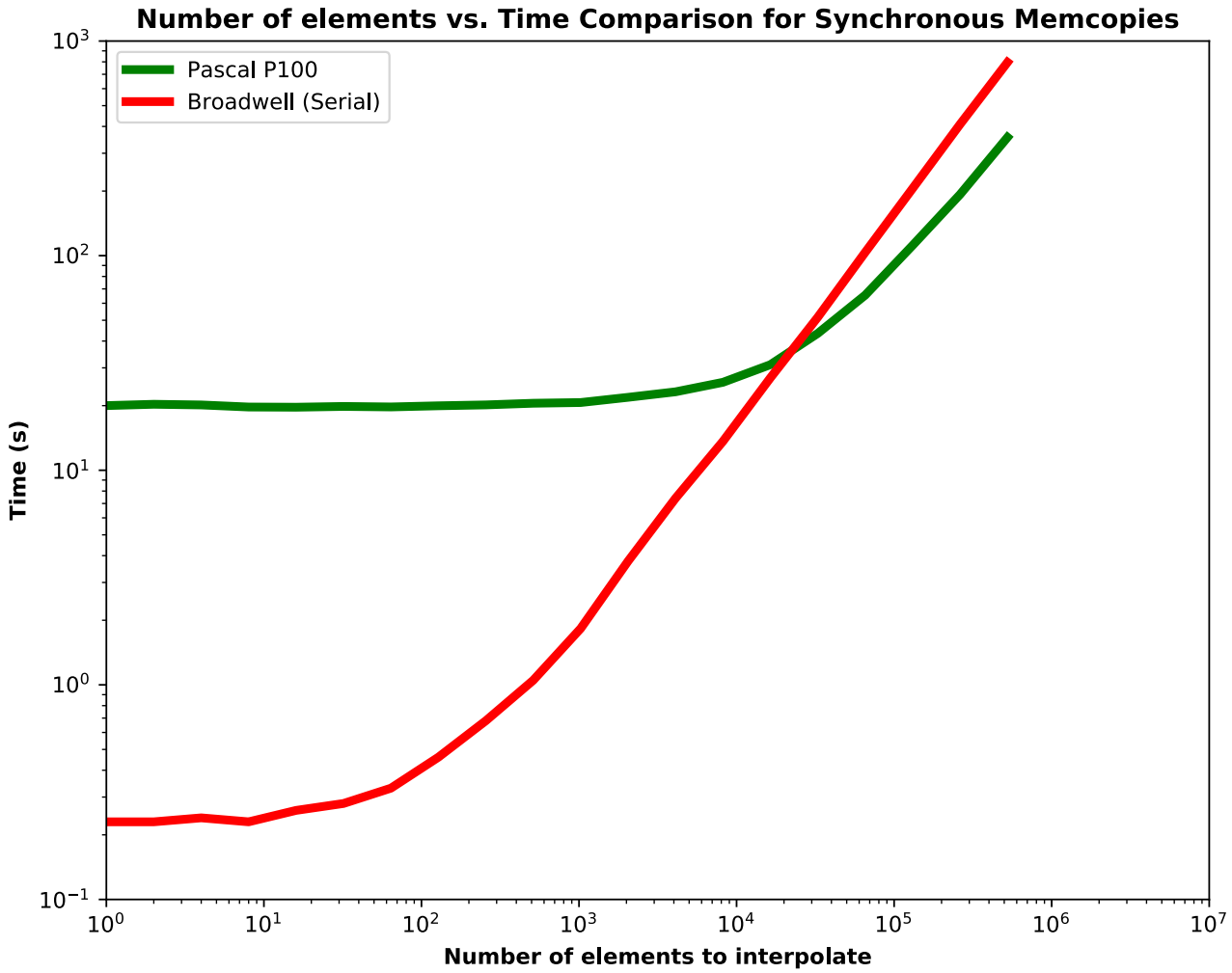


Bilinear Interpolation

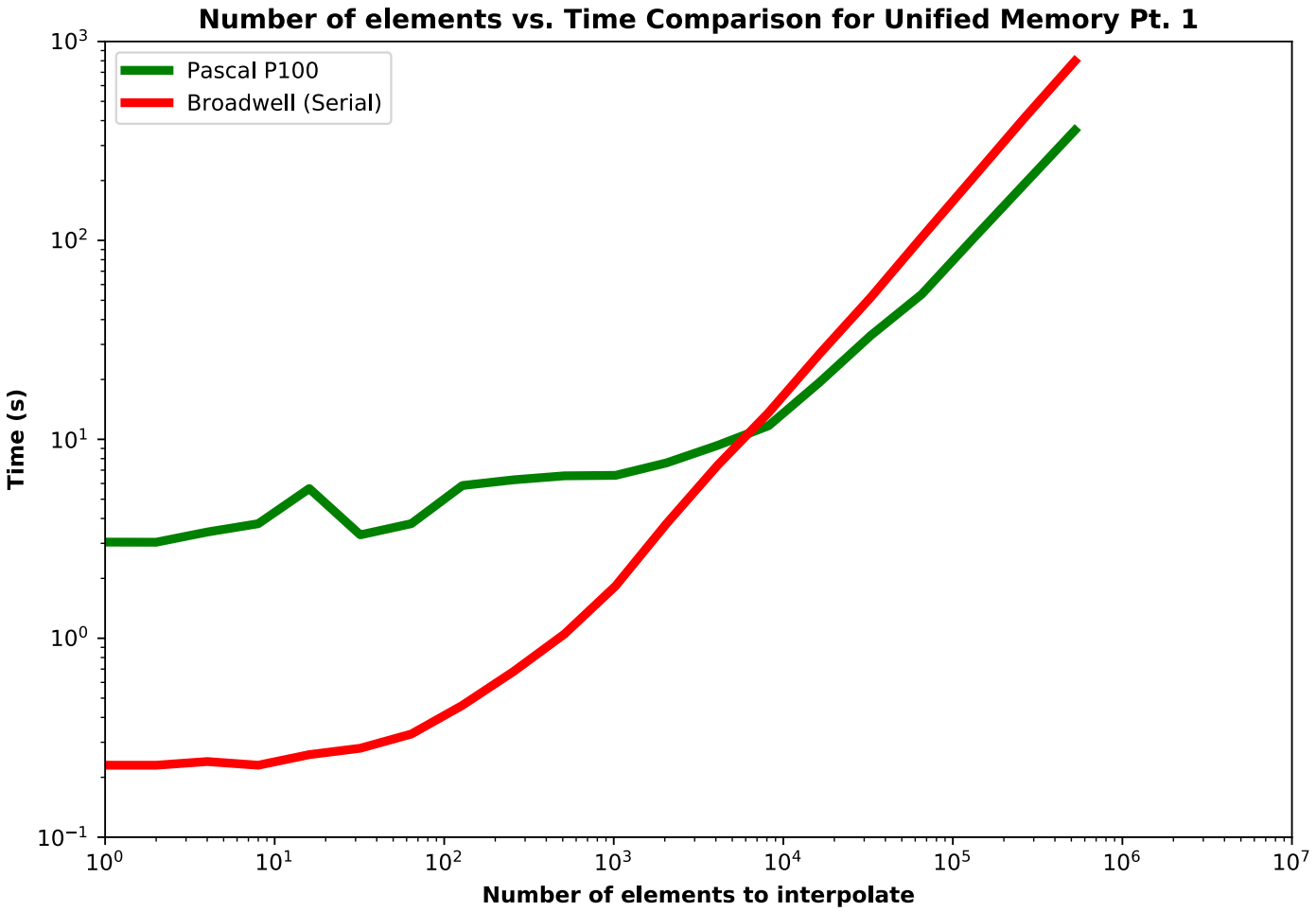
- Can be executed by the hardware using texture memory.
- Requires the use of 32 bit, single precision data types
- Use of single precision types provides an improved processing throughput of at least 2x, depending on architecture
- Texture memory allows us to store tables in the unified L1 cache/texture cache



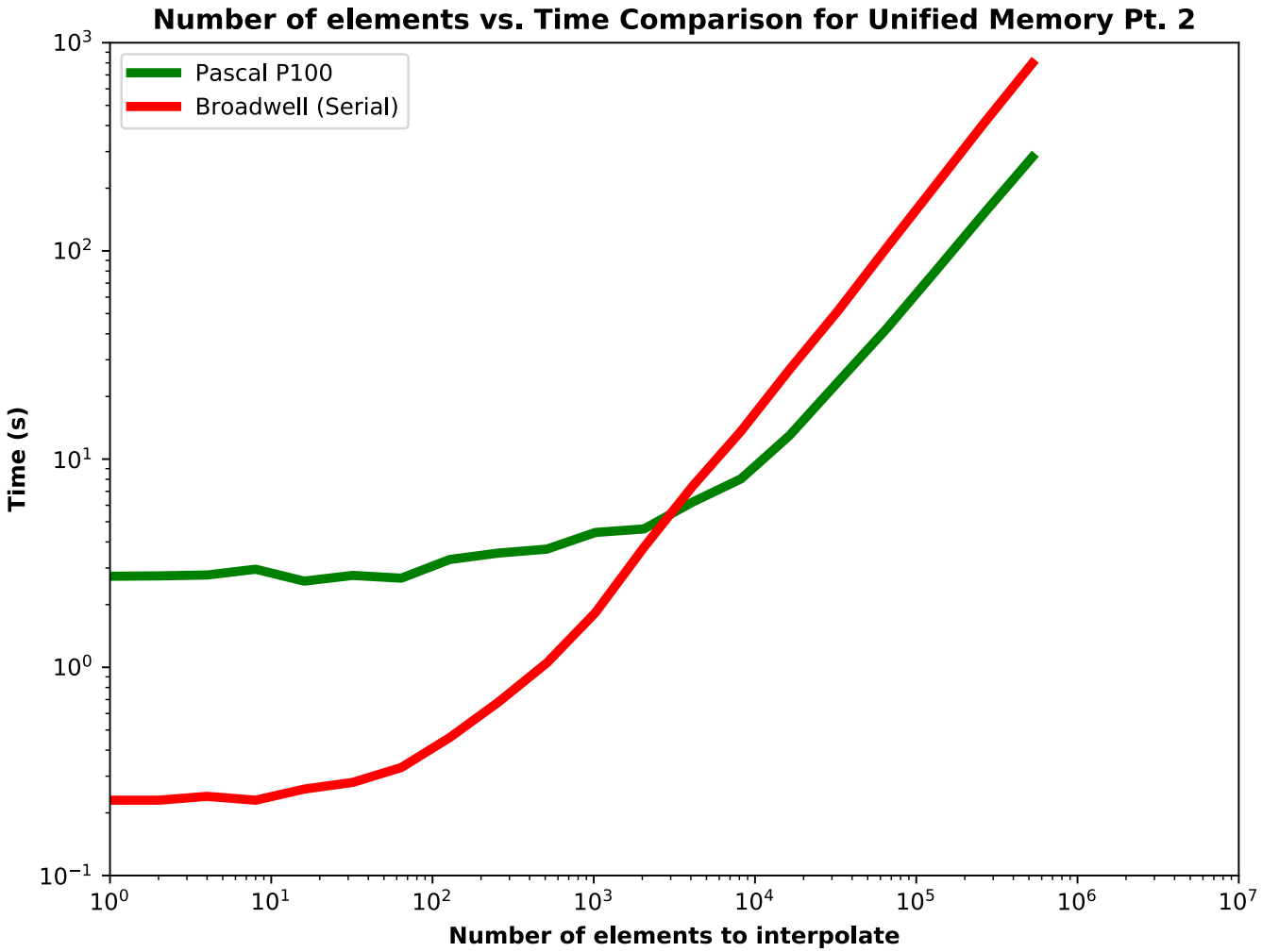
Speed-up from CUDA



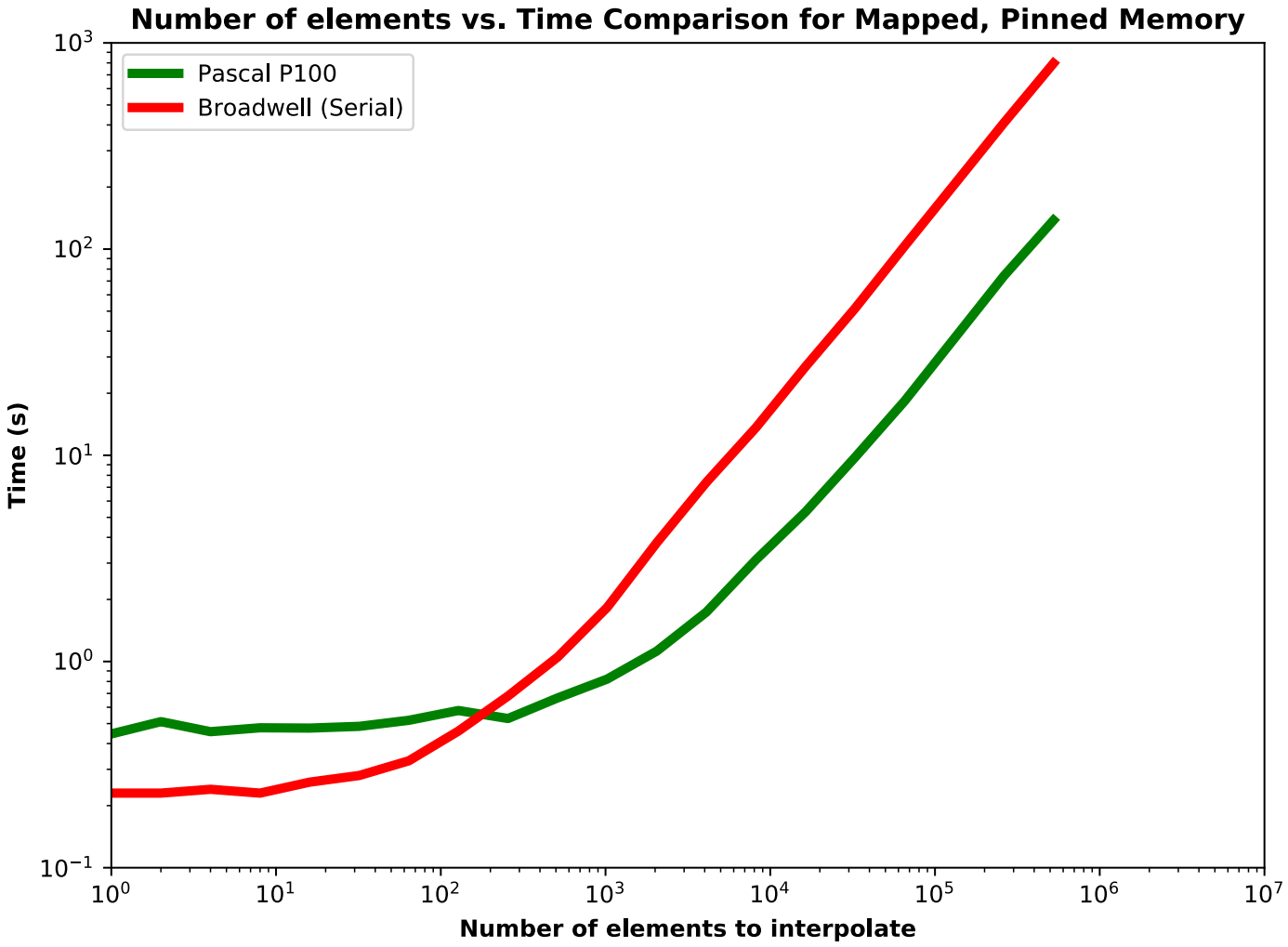
Speed-up from CUDA



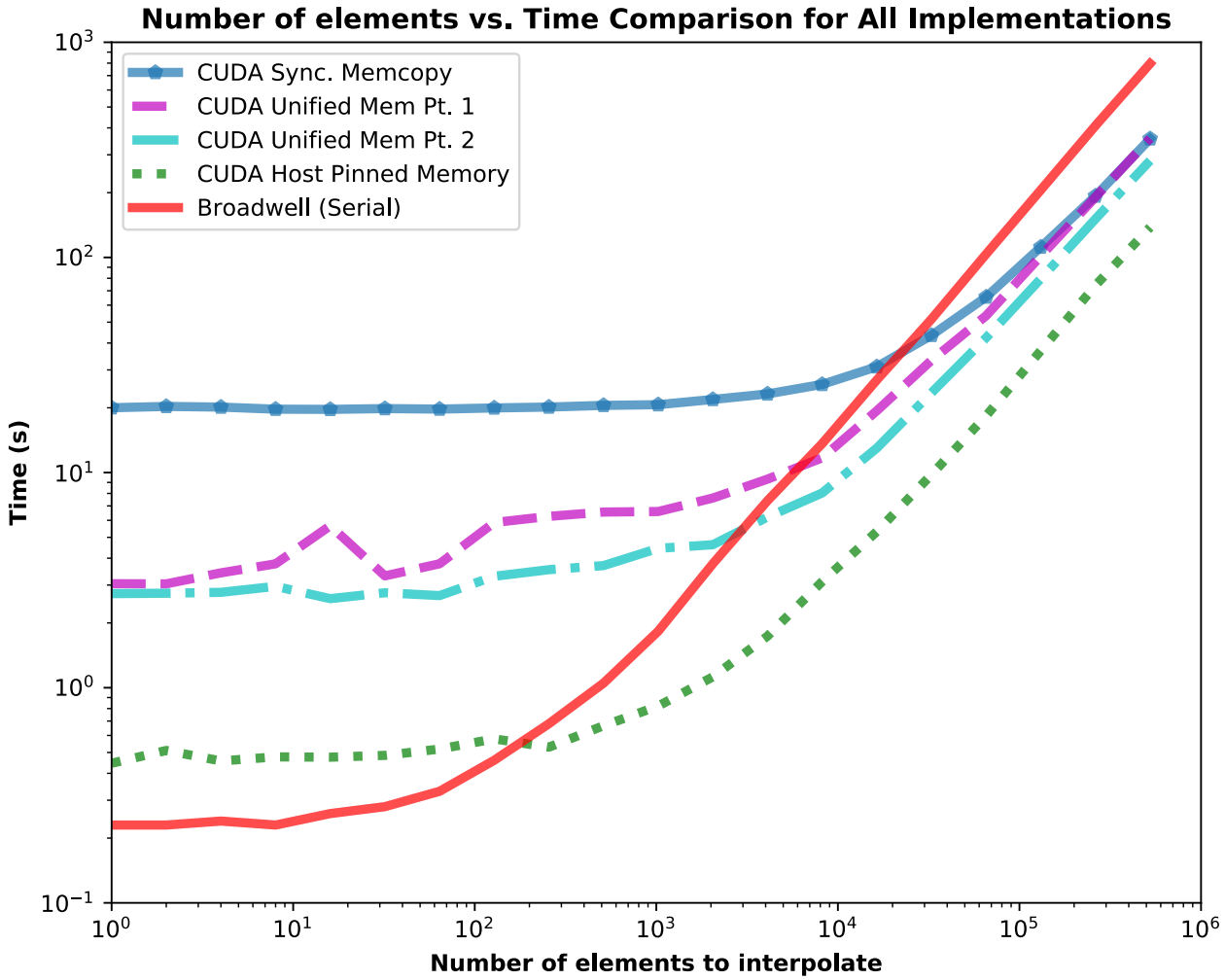
Speed-up from CUDA



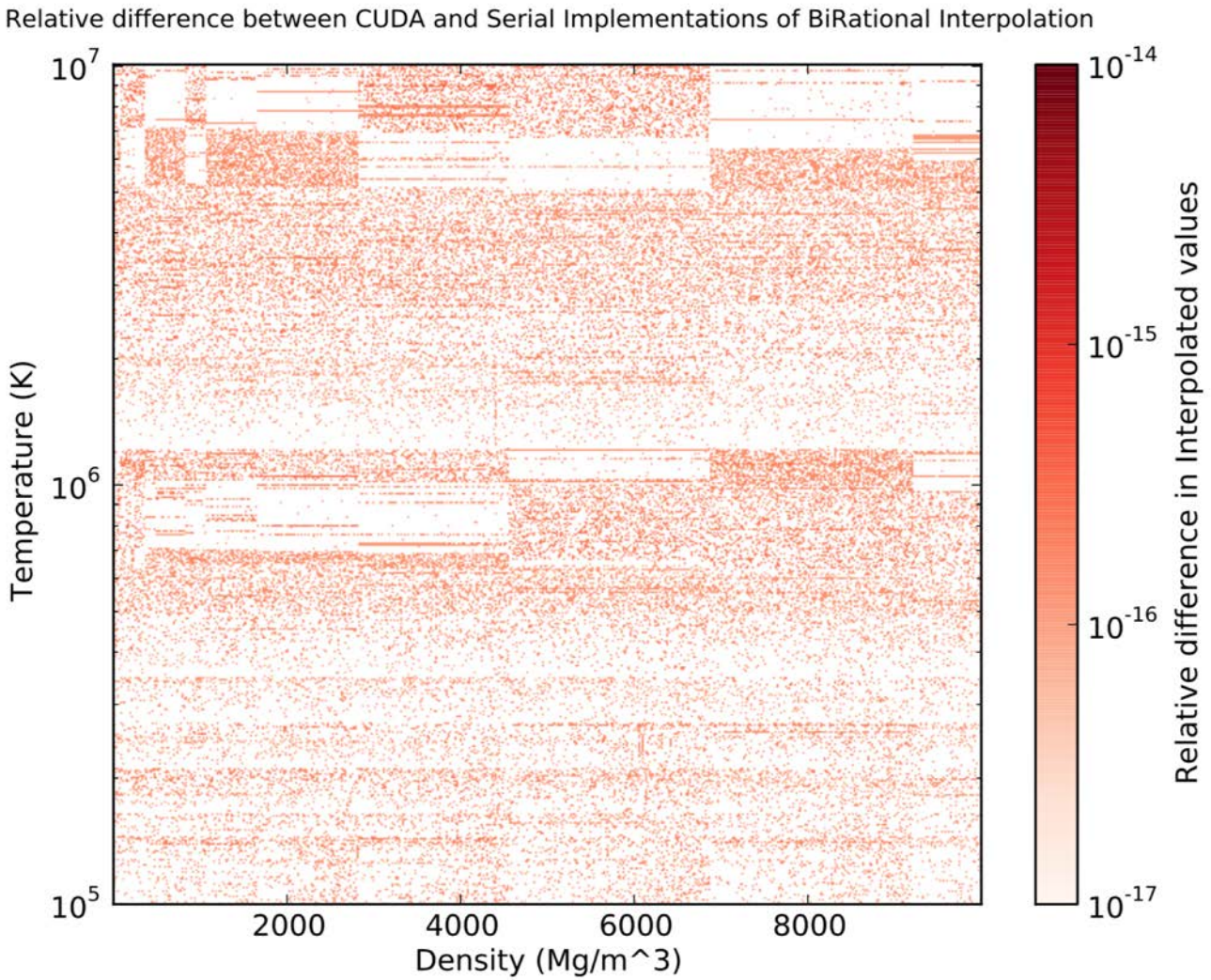
Speed-up from CUDA



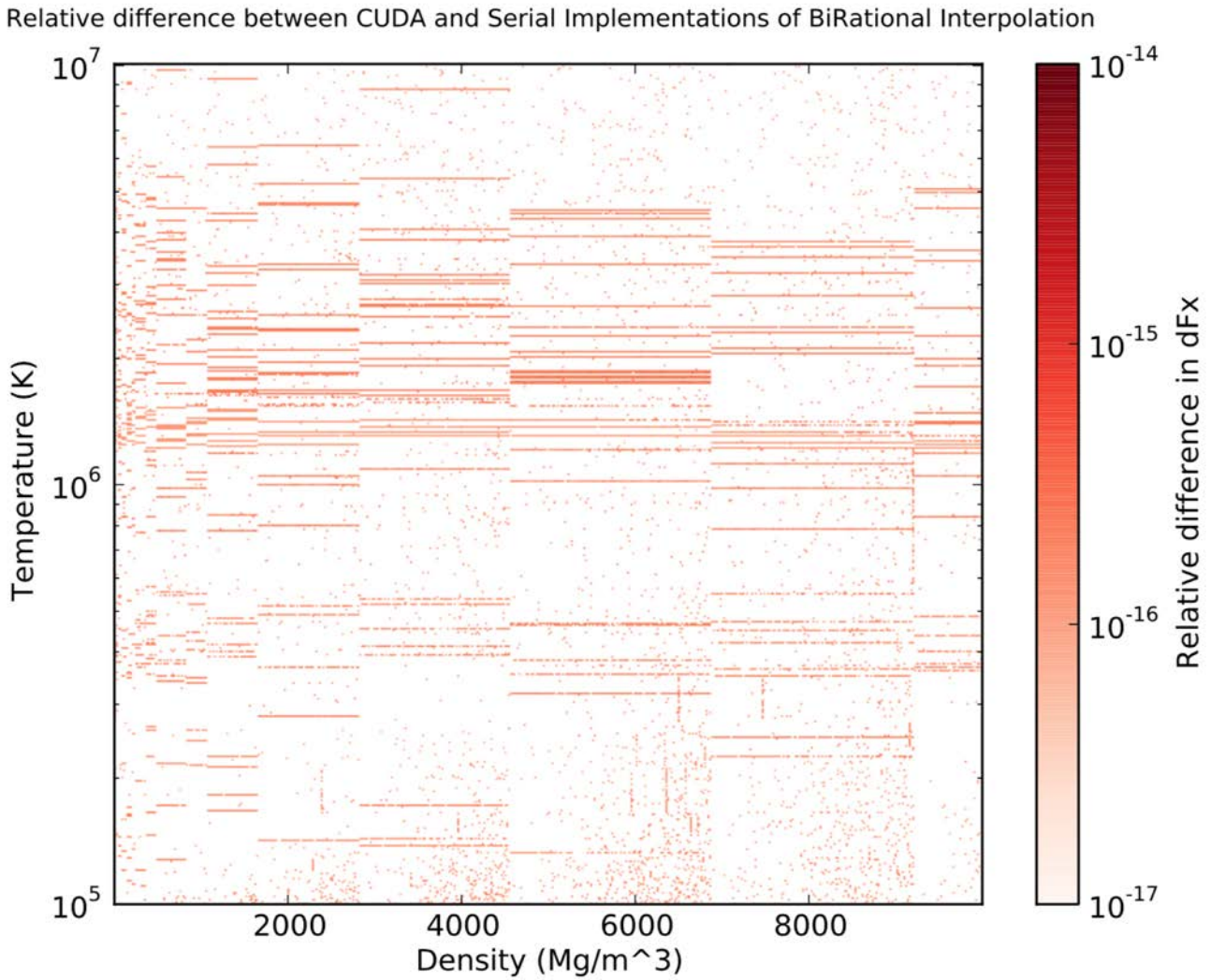
Speed-up from CUDA



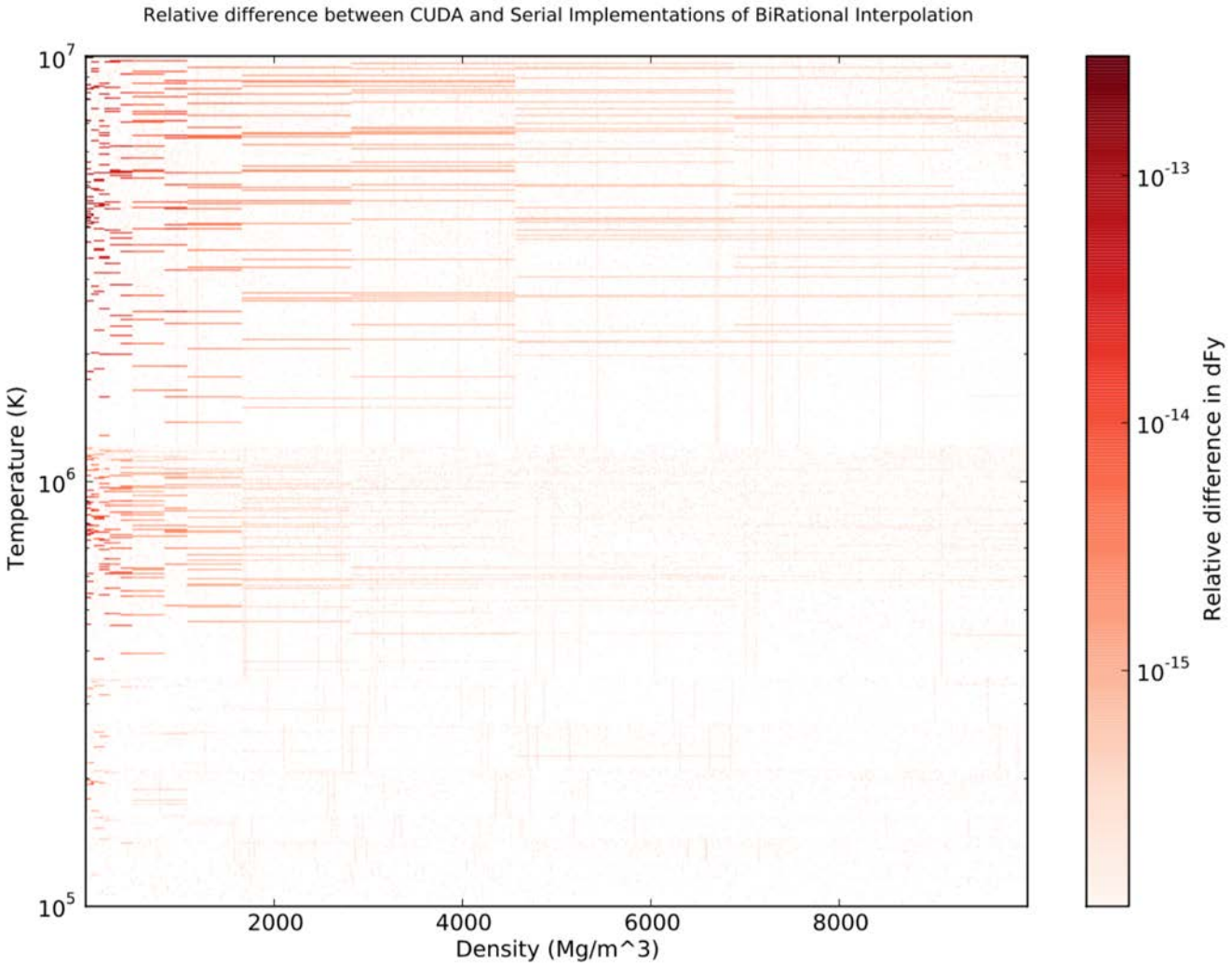
Accuracy for Sesame 3720



Accuracy for Sesame 3720



Accuracy for Sesame 3720



Machine Learning for EOS Interpolation

Machine Learning for EOS Interpolation

Goal: Replace calls to EOSPAC with ML regression.

Process:

1. Generate a training dataset using EOSPAC
2. Train the model
3. Characterize the error (compared to EOSPAC)
4. Integrate with FleCSALE

ML Models:

- Kernel Ridge Regression (KRR)
- Random Forest (RF)

Model Framework

Nomenclature: density (d), internal energy (ie), pressure (p), temperature (T), soundspeed (ss)

Pressure model

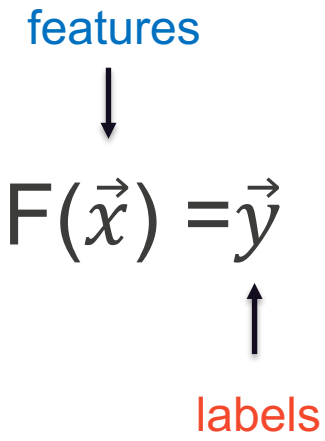
- F(d,ie) = p

Temperature model

- F(d,ie,p)=T

Soundspeed model

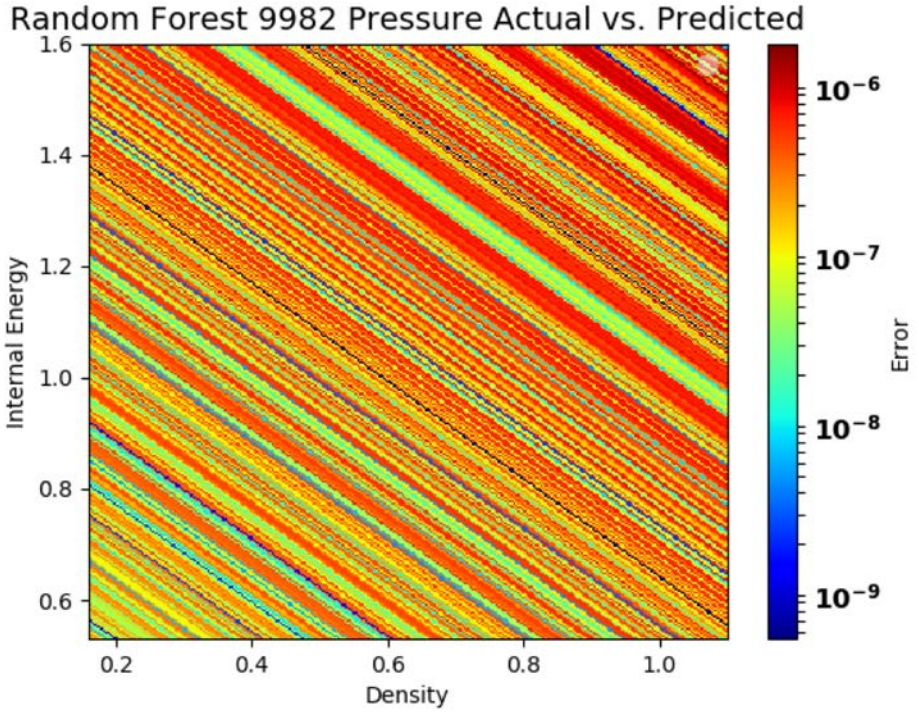
- F(d,ie,p,T) = ss

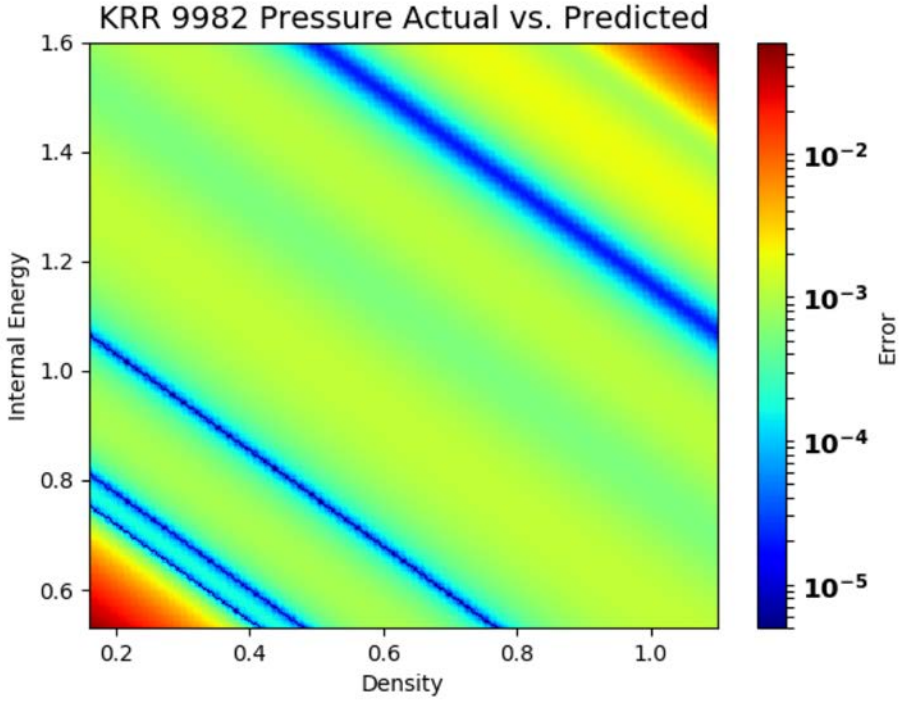


Machine Learning Error Analysis

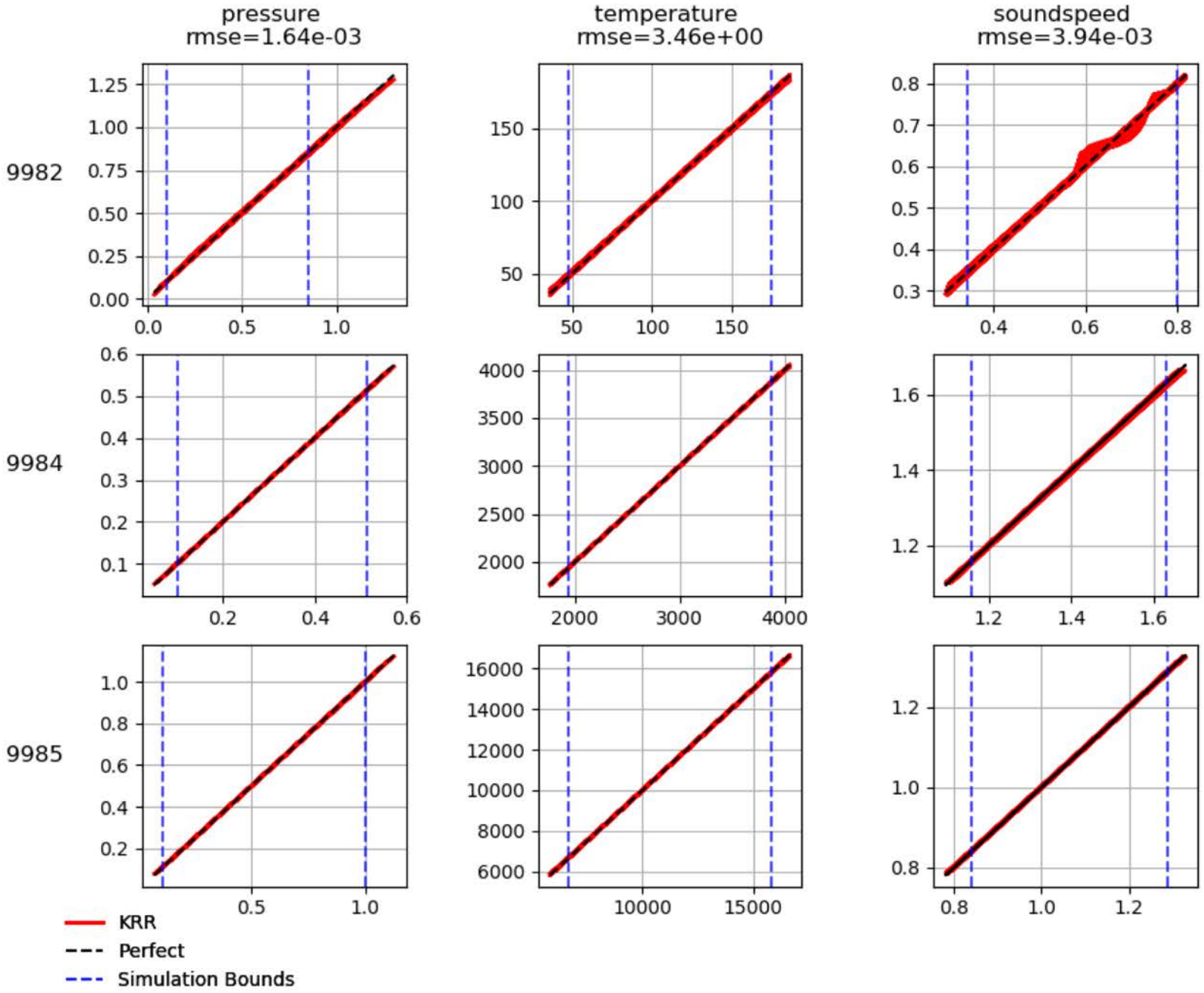
Random forest vs. Kernel Ridge Regression

- RF is more accurate but much slower
- RF takes ~3GB
- KRR takes ~1.5KB

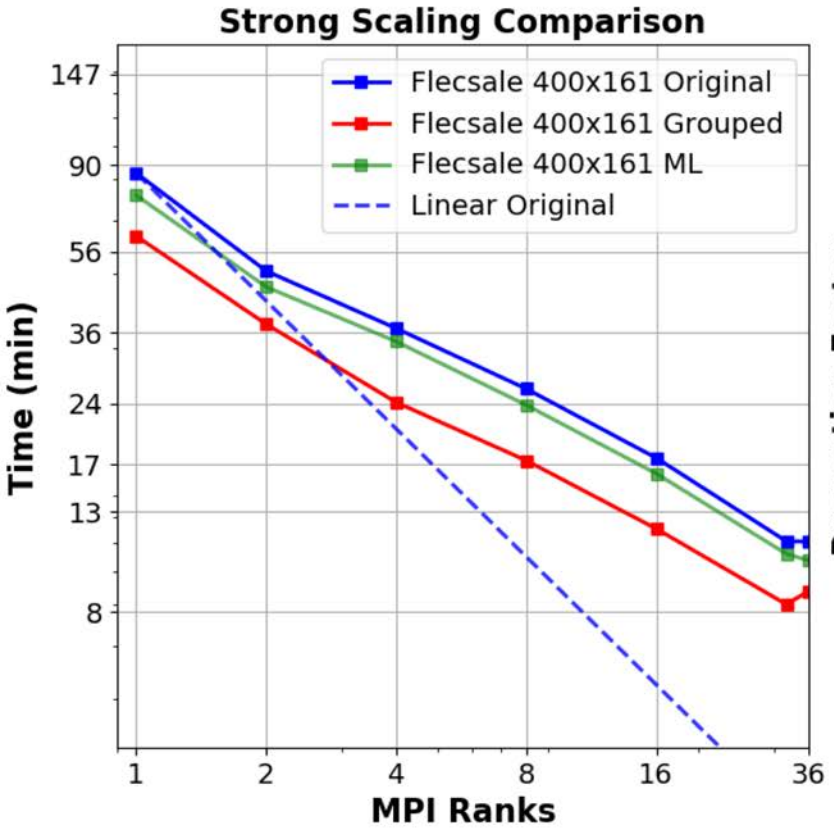


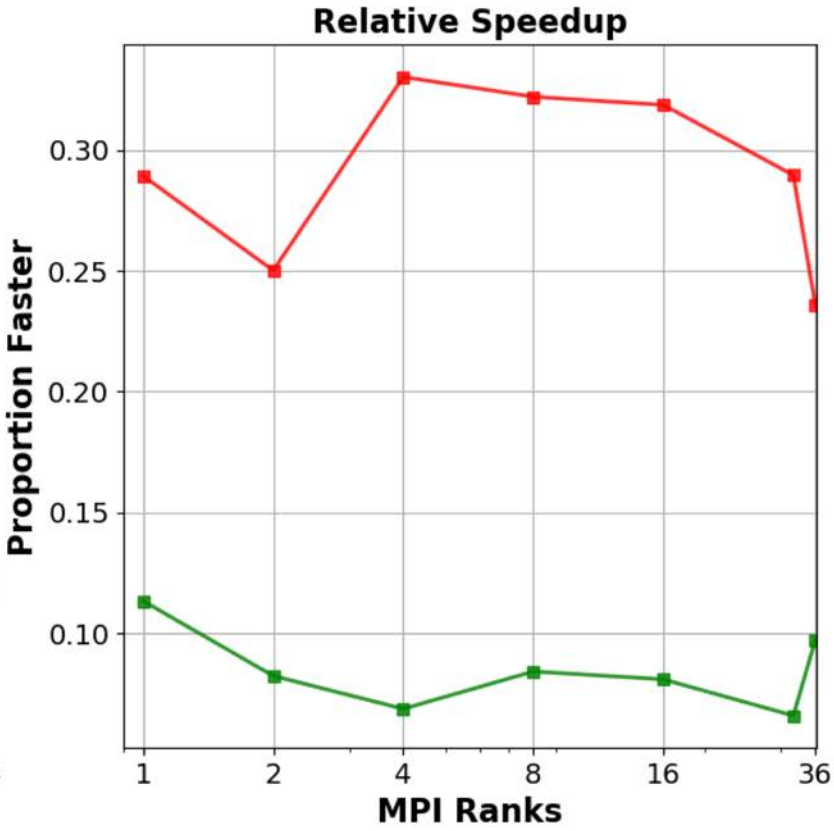


KRR Correlation plots



On-node Performance Comparison





Machine Learning Take-aways

- Accuracy and speed tradeoff
- Even with low accuracy, performance is questionable (in this case)
- Other ML models and C++ libraries exist

Final Results

Results

- **Proper selection of integration options and algorithms within EOSPAC provide a noticeable speed-up in FleCSALE**
- **Sparse data optimization with MPI backend shows a significant improvement about ~5x speed-up**
- **Initial CUDA results are promising and show a considerable speedup. However, integration with FleCSALE and host code optimizations are still required**
- **Initial machine learning integration did not increase performance, but other models and libraries could offer improvements**

Future Pursuits

- Add additional host code optimizations that allow for additional GPU performance improvements
- Take advantage of additional memory management schemes to further reduce the memory latency involved with GPU usage
- Study the performance of MPI and Legion backends, with and without the additional optimizations
- Utilize different parallel programming models within FleCSALE, (e.g., MPI + CUDA)
- Explore other types of Target Synchronization techniques for MPI communication like MPI_Lock, MPI_Unlock.
- Investigate other machine learning libraries (e.g., TensorFlow, MLPACK)

Acknowledgements

Thank you to the Co-Design Mentors, FleCSI Development Team, CCS-7, ISTI, ASC, & LANL